EV339073965US

Stream Cipher Cryptographic System and Method

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to Provisional Application No. 60/404,638 entitled "Stream Cipher Cryptographic System and Method" filed August 19, 2002, and assigned to the assignee hereof and hereby expressly incorporated by reference herein.

[0002] This application is related to the co-pending U.S. Application No. 09/846,443 filed on April 30, 2001 which is a continuation of U.S. Patent No. 6,252,958 filed on which is October 24, 1997 which is a continuation-in-part of U.S. Patent No. 6,510,228 filed on September 22, 1997, all entitled "Method and Apparatus For Generating Encryption Stream Ciphers" and assigned to the same assignee of the present application.

[0003] This application is also related to co-pending U.S. Application No. 10/145,365 filed on May 13, 2002 which is a continuation of U.S. Patent No. 6,490,357 filed on August 28, 1998, both entitled "Method and Apparatus For Generating Encryption Stream Ciphers" and assigned to the same assignee of the present application.

BACKGROUND

I. Field of Invention

[0004] The present invention relates to encryption and more particularly to a method and apparatus for generating encryption stream ciphers.

II. Description of the Related Art

[0005] Encryption is a process whereby data is manipulated by a random process such that the data is made unintelligible by all but the targeted recipient. One method of encryption for digitized data is through the use of stream ciphers. Stream ciphers work by taking the data to be encrypted and a stream of pseudo-random bits (or encryption bit stream) generated by an encryption algorithm and combining them, usually with the exclusive-or (XOR) operation. Decryption is simply the process of generating the same encryption bit stream and removing the encryption bit stream with the corresponding operation from the encrypted data. If the XOR operation was performed at the encryption side, the same XOR operation is also performed at the decryption side. For a secured encryption, the encryption bit stream must be computationally difficult to predict.

[0006] Many of the techniques used for generating the stream of pseudo-random numbers are based on linear feedback shift register (LFSR) over the Galois finite field of order 2. This is a special case of the Galois Finite field of order 2^n where n is a positive integer. For $n = 1$, the elements of the Galois field comprise bit values zero and one. The register is updated by shifting the bits over by one bit position and calculating a new output bit. The new bit is shifted into the register. For a Fibonacci register, the output bit is a linear function of the bits in the register. For a Galois register, many bits are updated in accordance with the output bit just shifted out from the register. Mathematically, the Fibonacci and Galois register architectures are equivalent.

[0007] The operations involved in generating the stream of pseudo-random numbers, namely the shifting and bit extraction, are efficient in hardware but inefficient in software or other implementations employing a general purpose processor or microprocessor. The inefficiency increases as the length of the shift register exceeds the length of the registers in the processor used to generate the stream. In addition, for $n = 0$, only one output bit is generated for each set of operations which, again, results in a very inefficient use of the processor.

[0008] An exemplary application which utilizes stream ciphers is wireless telephony. An exemplary wireless telephony communication system is a code division multiple access (CDMA) system. The operation of CDMA system is disclosed in U.S. Patent No. 4,901,307, entitled "SPREAD SPECTRUM MULTIPLE ACCESS COMMUNICATION SYSTEM USING SATELLITE OR TERRESTRIAL REPEATERS," assigned to the assignee of the present invention, and incorporated by reference herein. The CDMA system is further disclosed in U.S. Patent No. 5,103,459, entitled SYSTEM AND METHOD FOR GENERATING SIGNAL WAVEFORMS IN A CDMA CELLULAR TELEPHONE SYSTEM, assigned to the assignee of the present invention, and incorporated by reference herein. Another CDMA system includes the GLOBALSTAR communication system for world wide communication utilizing low earth orbiting satellites. Other wireless telephony systems include time division multiple access (TDMA) systems and frequency division multiple access (FDMA) systems. The CDMA systems can be designed to conform to the "TIA/EIA/IS-95

Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System", hereinafter referred to as the IS-95 standard. Similarly, the TDMA systems can be designed to conform to the TIA/EIA/IS-54 (TDMA) standard or to the European Global System for Mobile Communication (GSM) standard.

[0009] Encryption of digitized voice data in wireless telephony has been hampered by the lack of computational power in the remote station. This has led to weak encryption processes such as the Voice Privacy Mask used in the TDMA standard or to hardware generated stream ciphers such as the A5 cipher used in the GSM standard. The disadvantages of hardware based stream ciphers are the additional manufacturing cost of the hardware and the longer time and larger cost involved in the event the encryption process needs to be changed. Since many remote stations in wireless telephony systems and digital telephones comprise a microprocessor and memory, a stream cipher which is fast and uses little memory is well suited for these applications.

SUMMARY

[0010] Embodiments disclosed herein address the above stated needs by providing an efficient stream cipher which is fast and uses little memory.

[0011] In one aspect, a method of generating a key stream comprises applying a cryptographic function on input values selected from a first array of values to generate output values; selecting mask values from a second array of values; and combining the output values with the mask values to generate a key stream block for the key stream; wherein the first and second arrays are finite. The method may further comprise using a linear feedback shift register (LFSR) to generate the first array, wherein the values of the first array correspond to the values of the LSFR states. The second array may be generated by clocking the LFSR. The method may also further comprise applying the cryptographic function on updated input values selected from an updated first array of values to generate updated output values; selecting updated mask values from an updated second array of values; and combining the updated output values with the updated mask values to generate a new key stream block for the key stream. Here, the values of the first array may be set as the values of first LFSR states and the updated first array may be generated by clocking the LSFR. Similarly, the values of the second array may be set as the values of second LFSR states and the updated second array may be generated by clocking the LSFR

[0012] For a value comprising of one or more words, each comprising of two or more bytes, using the LFSR to generate the first array may comprise copying words of a key and words of an initialization vector into the LFSR; performing a byte-wise substitution on at least one byte of a word in the LFSR to generate a corresponding replacement word in the LFSR; mixing at least two bytes of a replacement word in the LFSR; and mixing at least two words in the LFSR to generate the first array.

[0013] Applying the cryptographic function may comprise performing a byte-wise substitution of at least one byte of an input value to generate primary intermediate values; and mixing at least two bytes of a primary intermediate value to generate a secondary intermediate value to generate the output values. The byte-wise substitution may comprise performing a nonlinear substitution of the at least one byte. The nonlinear substitution may be performed using a key-dependent Sbox substitution. The mixing of at least two bytes may comprise mixing at least two bytes using a minimum distance separable matrix. The key for the Sbox substitution may be generated based on a secret key of one or more words. The generation of the key may comprises performing a byte-wise substitution of at least one byte of a word of the secret key to generate a corresponding replacement word; and mixing at least two bytes of a replacement word to generate the first key byte.

[0014] Applying the cryptographic function may further comprise mixing at least two input values to generate the primary intermediate values. Applying the cryptographic function also may further comprise mixing at least two secondary intermediate values to generate the output values. Here, the mixing may be based on modular arithmetic.

[0015] In another aspect, apparatus for generating a key stream comprises means for applying a cryptographic function on input values selected from a first array of values to generate output values; means for selecting mask values from a second array of values; and means for combining the output values with the mask values to generate a key stream block for the key stream; wherein the first and second arrays are finite.

[0016] In still another aspect, a machine readable medium used for generating a key stream comprises code segment for applying a cryptographic function on input values selected from a first array of values to generate output values; code segment for selecting mask values from a second array of values; and code segment for combining the output values with the mask values to generate a key stream block for the key stream; wherein the first and second arrays are finite.

[0017] In a further aspect, apparatus for generating a key stream comprises a linear feedback shift register (LFSR) configured to generate a first array of values, wherein the values of the first array corresponds to the values of the LFSR states; a nonlinear filter module configured to apply a cryptographic function on input values selected from the first array to generate output values; and a combining module configured to combine the output values with mask values selected from a second array of values to generate a key stream block for the key stream; wherein the first and second arrays are finite.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] Various embodiments will be described in detail with reference to the following drawings in which like reference numerals refer to like elements, wherein:

[0019] Figure 1 shows an example implementation of the recurrence relation;

[0020] Figure 2 shows an example apparatus for generating a key stream;

[0021] Figure 3 shows an example method for generating a key stream;

[0022] Figure 4 shows an example of a fixed Sbox;

[0023] Figure 5 shows an example of a key dependent Sbox substitution;

[0024] Figure 6 shows an example apparatus for generating a key stream using five input words;

[0025] Figures 7A and 7B show a multiplication table using 0x72C688E8 for a linear feedback shift register;

[0026] Figure 8 shows an example key loading method; and

[0027] Figure 9 shows an example IV loading method.

DETAILED DESCRIPTION

[0028] Generally, embodiments describe a stream cipher for use in applications that places severe constraints on the amount of processing power, program space and/or memory available for software encryption algorithms. Voice encryption in wireless telephone is one example of such application. Since most mobile telephones in use incorporate a processor and memory, a software stream cipher that is fast and uses little memory would be ideal. In addition, many techniques used for generating the stream of pseudo-random bits are based on Linear Feedback Shift Registers (LFSRs) over the Galois Finite Field of order 2. However, such ciphers are difficult to implement efficiently on a general purpose processor. Accordingly, the present embodiments also overcome this dilemma by using an LFSR defined over GF($(2^8)^4$) (which is merely a different isomorphic representation of GF(2^{32})) in combination with a number of various techniques to greatly increase the generation speed of the pseudo-random stream in software on a general processor.

[0029] In the following description, specific details are given to provide a thorough understanding of the embodiments. However, it will be understood by one of ordinary skill in the art that the embodiments may be practiced without these specific detail. For example, circuits may be shown in block diagrams in order not to obscure the embodiments in unnecessary detail. In other instances, well-known circuits, structures and techniques may be shown in detail in order not to obscure the embodiments.

[0030] Also, it is noted that the embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed. A process may correspond to a method, a function, a procedure, a subroutine, a subprogram, etc. When a process corresponds to a function, its termination corresponds to a return of the function to the calling function or the main function.

I. Linear Feedback Shift Register

[0031] An LFSR is based on a recurrence relation over the Galois field. The output sequence in bits is defined by the following recurrence relation:

$$\boxed{S_{n+k} = C_{k-1}S_{n+k-1} + C_{k-2}S_{n+k-2} + \bullet\bullet\bullet + C_1S_{n+1} + C_0S_n} \quad (1)$$

where S_{n+k} is the output element, C_j are constant coefficients, k is the order of the recurrence relation, and n is an index in time. The state variables S and coefficients C are elements of the underlying finite field.

[0032] Figure 1 shows an example implementation of the recurrence relation in equation (1). For a recurrence relation of order *k*, registers 12a-12k comprise *k* elements S_n to S_{n+k-1}. The elements are provided to Galois field multipliers 14a-14k which multiply the elements with corresponding constants C_0-C_{k-1}. The resultant products from multipliers 14a-14k are provided to Galois field adders 16b-16k which sum the products to provide the output element. For $n = 1$, the elements of, for example Galois Field of order 2 (GF(2)) comprise a single bit (having a value of 0 or 1) and implementation of equation (1) requires many bit-wise operations. In this case, the implementation of the recurrence relation using a general purpose processor may not be efficient because a processor which is designed to manipulate byte or word sized objects is utilized to perform many operations on single bits.

[0033] More particularly, a LFSR may also be represented by its characteristic polynomial

$$p_1(x) = x^k + c_{k-1}x^{k-1} + c_{k-2}x^{k-2} + \ldots + c_1 x + c_0 \qquad (2)$$

where $p_1(x)$ indicates that multiplication and addition are over $GF(2^1)$. The operations involved, namely shifting and bit extraction, arc relatively efficient in hardware but inefficient in software, especially if the length of the shift register exceeds the length of the registers of the processor in question. In addition, only one bit of output is generated for each set of operations, which again is inefficient use of the general purpose processor.

[0034] However, LFSRs may operate over various finite fields and can be made more efficient in software by using a finite field more suited to a processor. In one embodiment, a Galois Field with 2^w elements ($GF(2^w)$), where *w* is related to the size of items in the underlying processor is used. Example sizes of items in a processor may be, but is not limited to, 16-bits or 2 bytes, 32-bit words and 64-bit words. Accordingly, the elements of this field and the coefficients of the recurrence relation may occupy one unit of storage and can be efficiently manipulated in software. In the meantime, the order *k* of the recurrence relation that encodes the same amount of state is reduced by a factor of *w*.

[0035] The field $GF(2^w)$ can also be represented as the modulo 2 coefficients of all polynomials with degree less than *w*. That is, an element *a* of the field is represented by a w-bit word with bits ($a_{w-1}, a_{w-2},\ldots,a_1, a_0$), which represents the polynomial in equation (3) below.

$$a_{w-1}x^{w-1} + a_{w-2}x^{w-2} + \ldots + a_1x + a_0 \quad (3)$$

The addition operation for such polynomials is simply addition modulo 2 (XOR) for each of the corresponding coefficients. The additive identity is the polynomial with all coefficients 0. Multiplication in the field is polynomial multiplication with modulo 2 coefficients, with the resulting polynomial being reduced modulo an irreducible polynomial of degree w. The multiplicative identity element is ($a_{w-1}, a_{w-2},\ldots,a_1, a_0$) = (0, 0, ..., 0, 1). Moreover, 8-bit bytes may be used to represent elements of GF(2^8) and 32-bit words may be used to represent degree-3 polynomials of bytes.

[0036] Accordingly, a LFSR may be specified in terms of bytes or words instead of bits, and successive output values will also be those units rather than bits. The feedback function is still of the form of equation (1), however the values s_i and the coefficients c_i are elements of $GF(2^w)$, rather than bits, and addition and multiplication are performed over $GF(2^w)$ as described above. Cycling or clocking the LFSR requires a number of constant multiplication operations followed by XOR of these terms. It is to be noted that multiplication by 0 simply means ignoring the corresponding element of the register, while multiplication by 1 requires no computation, as the output (of the multiplication) is the same as the input. Thus, to enable an efficient implementation, the multiplication constants, c_i, are chosen to be 0 or 1 most of the time.

[0037] The LFSR over the field $GF(2^w)$ is mathematically equivalent to w parallel shift registers over $GF(2)$, each of length equivalent to the total state w, each with the same recurrence relation but different initial state. Let the polynomial $p_1(x)$ represent the LFSR over $GF(2)$. This configuration makes updating a LFSR efficient when the LFSR has a maximum length period and when approximately half of the coefficients of $p_1(x)$ are 1. For example, the period has a maximum length of 2^{17w} when $p(x)$ is a primitive polynomial of degree 17.

[0038] Furthermore, the LFSR stream is not used in its entirety. To cycle the LFSR, elements $s_n,\ldots, s_{n+16}$ are used. After n cycles of the LFSR, the elements $s_n,\ldots, s_{n+16}$ are preserved and the values of these elements are called the state of the register. The state of the register corresponding to the elements $s_n,\ldots, s_{n+16}$ at any given instant in time is denoted $r_0, \ldots, r_{16}$. When the LFSR is cycled, the values in positions r_0, r_4, and r_{15} (s_n, s_{n+4} and s_{n+15}) are used to determine s_{n+17}. The values in positions $r_0, \ldots, r_{16}$ are shifted to the positions $r_0, \ldots, r_{15}$ respectively (that is, $r_l = r_{i+1}$, $15 \geq i \geq 0$) and r_{16} is set to the value of s_{n+17}. A nonlinear filter then uses the values in the register as input to the nonlinear function, as will be described below.

II. Apparatus implementing stream cipher in accordance with the present invention.

[0039] Figure 2 shows an example apparatus 200 implementing a stream cipher for generating a key stream. Apparatus 200 may comprise a LFSR 210, a nonlinear filter (NLF) module 220, a combining module 230 and a storage medium 240. LFSR 210 is configured to generate a first array of values, wherein the values of the first array corresponds to the values of the LFSR states. NLF module 220 is configured to apply a cryptographic function on input values selected from the first array to generate output values. Combining module 230 is configured to combine the output values with mask values selected from a second array of values. The resultant words output from combining module 230 may then be used to generate a key stream block for the key stream. Apparatus 200 may also comprise a processor 250 configured to control one or more of LFSR 210, NLF module 220, combining module 230 and storage medium 240. Here, the first and second arrays are finite.

[0040] It is to be noted that a value of an array may represent various bit units. However, for purpose of explanation, a bit unit of a word with one or more bytes will be used to describe the embodiments. In such case, LFSR 210 generates a first array of words, NLF module 220 applies a cryptographic function on input words selected from the first array to generate output words, and combining module 230 combines the output words with mask words selected from a second array of words to generate a key stream block for the key stream.

[0041] NLF module 220 may comprise a byte substitution module 222 configured to perform byte wise substitution of at least one byte of an input word to generate primary intermediate words, and a byte mixing module 224 configured to mix at least two bytes of a primary intermediate word to generate a secondary intermediate word. NLF module 220 may further comprise a word mixing module 226 and a word mixing module 228. Word mixing module 226 is configured to mix at least two input words based on modular arithmetic to generate the primary intermediate words. Word mixing module 228 is configured to mix at least two secondary intermediate words based on modular arithmetic to generate the output words.

[0042] In apparatus 200, a portion or more of LFSR 210, NLF module 220, combining module 230 and storage medium 270 may be implemented together. For example, word mixing modules 226 and 228 may be implemented together. Moreover, a portion or more of apparatus 200 may be implemented by hardware, software, firmware, middleware, microcode, or any combination thereof. When implemented in software, firmware, middleware or microcode, the program code or code segments to perform the necessary tasks may be stored in a machine readable medium such as storage medium 240 or in a separate storage(s) not shown. A processor such as processor 250 may perform the necessary tasks. A code segment may represent an instruction, procedure, a function, a subprogram, a program, a routine, a subroutine, a module, a software package, a class, or any combination of instructions, data structures, or program statements. A code segment may be coupled to another code segment or a hardware circuit by passing and/or receiving information, data, arguments, parameters, or memory contents. Information, arguments, parameters, data, etc. may be passed, forwarded, or transmitted via any suitable means including memory sharing, message passing, token passing, network transmission, etc.

[0043] Furthermore, it should be apparent to those skilled in the art that the elements of apparatus 200 may be rearranged without affecting the operation. The details of apparatus 200 will be described with reference to Figure 3 below.

[0044] Figure 3 shows a method 300 for generating a key stream. To generate the key stream, a cryptographic function is applied on input values selected from a first array of values to generate output values (310). Mask values are selected from a second array of values (320) and the output values are combined, i.e. XOR operated, with the mask values (330). Here, the first and second arrays are finite. The number of input values and the number of output values are also typically equal. In addition, the second array may be generated from the first array, and each of the first and/or second array may comprise seventeen values.

[0045] The resultant combination of the output values and the mask values are then used to generate a key stream block for the key stream. Multiple key stream blocks may be generated as necessary by repeating process 310 to 330. For example, a cryptographic function may be applied on updated or new input values selected from an updated first array of values to generate updated output values, updated mask values may be selected from an updated second array of values, and the updated output values may then be combined with the updated mask values. In method 300, LFSR 210 may be used to generate either one or both the first array and the updated first array, wherein the values of the first array correspond to the values of LSFR states. The second array and/or updated second array may be generated by cycling or clocking LFSR 210. Alternatively, the values of the second array may be set as the values of LFSR states and the updated second array may be generated by clocking LSFR 210.

[0046] Assuming that the unit of a value in an array is a word, applying the cryptographic function may include performing a byte-wise substitution of at least one byte of an input word to generate primary intermediate words, and mixing at least two bytes of a primary intermediate word to generate a secondary intermediate words to generate the output words. Here, byte substitution module 222 may perform the byte-wise substitution and byte mixing module 224 may perform the byte mixing.

[0047] More particularly, the byte-wise substitution of a byte may comprise performing a nonlinear substitution of the byte. For example, a key-dependent Sbox substitution may be performed on the byte. S-box is a term for a nonlinear function often implemented as a table lookup. These S-boxes are based in turn on a fixed $m \times m$ permutation S-box, iterated with the data modified by variables derived during a secret key setup. A fixed S-box will be referred to as *Sbox[.]*. An example of a fixed 8x8 permutation S-box having a nonlinearity of 104 is shown in Figure 4.

[0048] Key-dependent Sbox substitution generally involves combining the byte to be substituted with a first key to generate a first combined byte and substituting the first combined byte with a byte value from a pre-determined or pre-calculated array. Figure 5 shows an example of a key-dependent Sbox substitution. As shown, a byte to be substitute e_1 is combined or XOR operated with a first key byte k_1. The resultant byte o_1 serves as an index used to select a byte value from the Sbox that will substitute o_1. The replacement byte e_2 is combined with a second key byte k_2 and the resultant o_2 is substituted with a byte value from the Sbox as described. The combining and substitution may be repeated as necessary. The key bytes are derived from a secret key stream and will be discussed below in Section V.

[0049] The mixing of at least two bytes of the primary intermediate words may comprise mixing at least two bytes using a minimum distance separable (MDS) matrix multiplication. Referring to Figure 2, the mixing may be performed by byte mixing module 224. Also, the minimum distance separable matrix multiplication typically comprises operations over a Galois Field comprising 256 elements. An example MDS matrix is discussed below in Section IV.

[0050] Applying the cryptographic function in process 310 may further comprise mixing at least two input words to generate the primary intermediate values. Similarly, applying the cryptographic function may further comprise mixing at least two secondary intermediate words to generate the output values. The mixing may be based on modular arithmetic. More particularly, the mixing of the input words may comprise adding selected input words to generate a mixed word and adding the mixed word with each input word to generate the primary intermediate words. Word mixing module 226 may perform the mixing of the input words. The mixing of secondary intermediate words may comprise adding the secondary intermediate words to generate a mixed word and adding the mixed word with each secondary intermediate word to generate the output words. Word mixing module 228 may perform the mixing of the secondary intermediate words. In some embodiments, a Pseudo Hadamard Transform may be used to mix the input items as described below.

IV. Example embodiment implementing a stream cipher

[0051] Figure 6 shows an example embodiment 600 for generating a key stream using five selected input words. Embodiment 600 involves mixing words {A, B, C, D, E} selected from a LFSR 610 by a Pseudo-Hadamard Transform 620, substitution of bytes by a keyed S-box substitution 630, mixing of words by a MDS matrix multiplication 640, mixing of the words again by a Pseudo-Hadamard Transform 650, and combining the original input words by XOR 660. The component of embodiment 600 that is nonlinear is S-box substitution 630. However, additional nonlinearity also comes from the combination of the operations of addition modulo 2^{32} and XOR 660 because while each of these operations is linear in its respective mathematical group, each is nonlinear in the other's group.

[0052] Assume β, bytes, represents the Galois finite field GF(2^8) represented modulo the irreducible polynomial $z^8 + \gamma_7 z^7 + \gamma_6 z^6 + \gamma_5 z^5 + \gamma_4 z^4 + \gamma_3 z^3 + \gamma_2 z^2 + \gamma_1 z + \gamma_0$, where the γ_i are bits, specifically $z^8 + z^6 + z^3 + z^2 + 1$. The constant β_0 = 0xE8 represents the polynomial $z^7 + z^6 + z^5 + z^3$ for example. Also, assume W, words, represents the Galois finite field GF(B^4) represented modulo the irreducible polynomial $y^4 + \beta_3 y^3 + \beta_2 y^2 + \beta_1 y$

$+ \beta_0$, where the $\beta_i \in B$ are bytes, specifically $y^4 + 0x72.y^3 + 0xC6.y^2 + 0x88.y + 0xE8$. The element α used below is the polynomial y. LFSR 610, then may comprise of 17 32-bit words, with characteristic polynomial $x^{17} + x^{15} + x^4 + \alpha$, where $\alpha \in W$ is the polynomial y.

[0053] Having defined such relation, LFSR 610 may be implemented more efficiently because the constant α is relatively simple. Multiplication by α involves shifting the word left by 8 bits, and adding (XOR) a precomputed constant from a table indexed by the most significant 8 bits. For example, in the computer language C, calculating a new word to be inserted in LFSR 610 is as follows.

```
new = R[15] ^ R[4] ^ (R[0] << 8) ^ Multab[R[0] >> 24];
with the precomputed table:
        /* Multiplication table using 0x72C688E8 */
        unsigned long Multab[256] shown in Figures 7A and 7B.
```

LFSR 610 can then be updated if necessary as follows.

```
R[0] = R[1]; R[1] = R[2]; ... ; R[16] = R[17]; R[17] = new;
```

[0054] After updating LFSR 610, five words from the LFSR states are selected for input to the nonlinear filtering process. These are r_{17}, r_{13}, r_6, r_1, r_0, referred to as A, B, C, D, E. These tap positions form a "full positive difference set," such that as words move through the register and are selected as input to the nonlinear filtering process, no pair of words is used more than once.

[0055] The selected input words {A, B, C, D, E} are mixed by Pseudo Hadamard Transform 620. More particularly, the operation mixes words, using addition modulo 2^{32}, and further mixes an arbitrary number of words. For example, the five words may be mixed by the matrix multiplication as follows.

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix}$$

[0056] In the above matrix multiplication, all diagonal entries are 2 except the last diagonal entry is 1. This may for example be implemented in a program language as follows:

```
E = A + B + C + D + E;
A += E; B += E; C += E; D += E;
```

[0057] After mixing words as described, a keyed sbox substitution 430 is performed. In one embodiment, four logically independent S-boxes may be used. In such case, sbox substitution would involve four rounds of combination and substitution as described with reference to Figure 5. More particularly, the four keyed S-boxes S_i $(0 \leq i \leq 3)$ may be derived from a fixed one in the following manner:

[0058] $S_i(x) = \text{Sbox}[K_{i,N-1} \oplus \text{Sbox}[K_{i,N-2} \oplus \ldots \text{Sbox}[K_{i,0} \oplus x]\ldots]]$

where $\oplus$ is the XOR operator, and N is the number of words in a key. The byte values $K_{i,j}$ are derived from the secret key in a manner described below.

[0059] After the bytes have been processed by the keyed S-boxes 640, the four bytes in each word are mixed by MDS matrix 650 multiplication. The bytes of a word and of the MDS matrix are treated as elements of B. An example of the MDS matrix is as follows:

```
unsigned char MDS[4][4] = {
{ 2, 3, 1, 1 },
{ 1, 2, 3, 1 },
{ 1, 1, 2, 3 },
{ 3, 1, 1, 2 }
}
```

[0060] The values "2" represent the polynomial "z", and "3" represents the polynomial "$z + 1$". The resulting words are again mixed by Pseudo Hadamard Transform 650. A key stream block is then generated by combining the mixed words with another five input words {A', B', C', D', E'}. Here, A', B', C', D', E' may be new input words selected from LFSR 610. Alternatively, A', B', C', D', E' may be new input words selected from an updated LFSR 610 by clocking LFSR 610.

[0061] The first four operations of the nonlinear filter are invertible. In particular, the effect of the MDS matrix multiplication and the second PHT stage are unkeyed and may be known to an attacker. XORing the input words to the output has two effects. First, it makes the output not a strict permutation of the input, that is, a pseudo-random function instead of a pseudo-random permutation. Second, these XOR operations "lock in" the mixing of the two stages mentioned above, since an attacker needs to remove the effects of these words before being able to reverse these mixing rounds.

[0062] The five words produced by this processing are used as the output keystream. The order $\{A..E\}$ may be from most significant byte of each word first. Accordingly, a key stream block may be generated.

[0063] It is to be noted that if sufficient storage medium is available, the operations of the four keyed S-boxes and the subsequent MDS matrix multiplication can be pre-calculated at the time of key setup, resulting in four tables, one for each byte of the input word. The output word is calculated by XORing the 4 32-bit table entries thus selected. Since many current high-end processors allow multiple instructions to execute at once, if the instructions are sufficiently independent. Note also that the operations in embodiment 600 may be performed in parallel, allowing very good performance on processors. Moreover, similar to apparatus 200, a portion or more of embodiment 600 may be implemented by hardware, software, firmware, middleware, microcode, or any combination thereof.

V. Key Loading and Initialization Vector

[0064] Generation of a key stream as described above may also involve secret key and Initialization Vector (IV). For purposes of explanation, a bit unit or a word with two or more bytes will be used to describe both key loading and IV loading process. In such case, a secret key and IV may be presented as a byte stream and may be converted to 32-bit words in the most significant byte first (big-endian) representation. The secret key and IV are then multiples of 4 bytes each. The minimum size of the key may be 32 bits while the maximum key size may be set to, for example, 320 bits. The minimum size of the IV may be zero while the maximum size of the IV is determined by the key length, such that the sum of the key length and IV length is no more than, for example, 12 words (384 bits). The size of the IV may also vary. Different key/IV length combinations will generate distinct output streams. No more than 2^{64} (160-bit) blocks of output should be generated using any one key/IV combination.

[0065] Figure 8 shows an example key loading method 800 for a secret key of one or more words. In method 800, a byte-wise substitution of at least one byte of a key word is performed to generate replacement words (810). At least two bytes of a replacement word is then mixed to generate an encrypted key (820). Here, the byte-wise substitution may be performed by byte substitution module 222 or by a separate module (not shown). Similarly, the byte mixing may be performed by byte mixing module 224 or by a separate module (not shown). Also, the original secret key and the encrypted key may be stored in storage medium 240.

[0066] More particularly, key loading process mixes the key bytes through a fixed S-box and MDS processes to ensure that all or nearly all bytes of the key affect four of the keyed S-boxes. For each word of the key, the bytes are substituted through the fixed S-box, then the resulting word is mixed by the MDS matrix multiplication. The resulting words or the encrypted words are stored for subsequent use as described above. The resulting words may occupy the same amount of space as the original key, which is no longer needed. Namely, these resulting words are used in the key-dependent S-box substitution as described with reference to Figure 5, and also during the IV loading process as described below to initialize a LFSR. The bytes $K_{i,j}$ discussed above are the bytes of these stored words; the j index ($0 \leq j < N$, where N is the number of words of the key) locates the word of the stored mixed key, while the i index ($0 \leq i \leq 3$) is the byte of the word, with the byte numbered 3 being the most significant byte.

[0067] If the faster implementation is desired, the effect of the keyed S-boxes followed by the MDS matrix multiplication can be computed into four tables, each with 256 32-bit entries. The combined operations then consist of four byte-index table lookups and four word XOR operations. Also, key-loading stage may be implemented as described or by various other known techniques. Alternatively, for embedded applications with a secret key fixed in read-only memory, a high quality cryptographic key may be used directly as if it was the output of the key loading stage.

[0068] The IV loading process initializes a LFSR with values derived from a non-linear mixing of the key and the IV. Figure 9 shows an example IV loading method 900. For purposes of explanation, let L be the length (in words) of the key, and I be the length in words of the IV. The encrypted key, generated by above by key loading, or the original key, if there no key loading is performed, is copied into the first L words of the LFSR

(910). The L words is then followed by the I words of the IV (920). A byte-wise substitution is performed on the words in the LFSR to generate replacement words (930) and at least two bytes of a replacement word is mixed to generate initial LFSR words (940). Two initial words may then be mixed to generate an array of words from input words are selected (950). Here, the byte-wise substitution may be performed by byte substitution module 222 or by a separate module (not shown). Similarly, the byte mixing may be performed by byte mixing module 224 or by a separate module (not shown), and the initial word mixing may be performed by either word mixing module 226 or 228 or by a separate module (not shown). Also, the IV may be stored in storage medium 240.

[0069] More particularly, the words are processed using the fixed S-box and MDS matrix multiplication as in the key loading process above. (Note that the fixed S-box is used to avoid undesirable interactions since the keyed S-boxes use the same key bytes.) For example, for a LFSR having 17 states, the next word of the shift register is initialized based on L and I, as follows:

$$R[L+I] = ((L << 4) + I) * 0x01010101;$$

That is, each byte of that word is initialized to $(16L + I)$. This word is passed through the keyed S-boxes and MDS matrix multiplication as would be done in normal operation of the cipher. Generally, this will result in distinct values of the bytes of this word.

[0070] The remaining $(17\text{-}L\text{-}I\text{-}1)$ words of the register are filled by adding the word immediately previous word $(L+I)$, then processing the resulting word with the keyed S-boxes and MDS matrix multiplication. For example, if L is 4 (a 128-bit key) and I is 2 (64-bit IV), r_6 would be filled with the value 0x42424242 before being processed by the keyed S-boxes and MDS matrix multiply. Then r_7 would be set to $r_6 + r_0$, and mixed by the keyed S-boxes and MDS matrix multiply. Similarly r_8 would be set to $r_7 + r_1$, and mixed by the keyed S-boxes and MDS matrix multiply, and so on until r_{16} is set to $r_{15} + r_9$ and mixed.

[0071] Once the LFSR has been filled with data, the contents may be mixed with a 17-word-wide PHT. Key stream generation can then begin as described above.

VI. Conclusion

[0072] Therefore, an efficient stream cipher is disclosed. More particularly, the stream cipher combines an LFSR generator with a keyed mixing function reminiscent of a block cipher round. Also, embodiments use native processor operations on word-sized data items, but is expected to accept keys which are simply strings of bytes, and to produce a stream of bytes as output for encryption purposes. This means that a translation between native byte ordering and external byte ordering may be needed to ensure compatibility between implementations running on different processors. Since most Internet standards are defined using "big-endian" byte ordering, in which the most significant byte of a multi-byte quantity appears first in memory, embodiments may use this "big-endian" byte ordering. On "little-endian" machines, the bytes of the key and IV must be assembled into words, and the words of the output stream must be byte reversed before being XORed into the buffer.

[0073] It should be noted that the foregoing embodiments are merely examples and are not to be construed as limiting the invention. The description of the embodiments is intended to be illustrative, and not to limit the scope of the claims. As such, the present teachings can be readily applied to other types of apparatuses and many alternatives, modifications, and variations will be apparent to those skilled in the art.

What is claimed is: